University of Victoria

Department of Electrical and Computer Engineering

CENG/ELEC/SENG 499

MAY – AUGUST 2010

Final Report

Project Number: #02

Project Title: SNA2 – Wireless Distribution of Electric Loads

Date: 30 July, 2010

Submitted To: Dr. Subhasis Nandi

Names:

Karl Fort V00243637 (kfort@uvic.ca)
Danish Qureshi V00174865 (danishr@uvic.ca)
Tom Nute V00219861 (tomnute@uvic.ca)
Steve Smith V00215982 (smiths@uvic.ca)

Dr. Subhasis Nandi
Associate Professor
Electrical and Computer Engineering
University of Victoria
P.O. Box 1700
Victoria, B.C.
V8W 2Y2

July 30, 2010

Dear Dr. Nandi,

Please accept the accompanying report entitled "Wireless Distribution of Electric Loads" submitted as a required component of the ELEC/SENG 499 course for May – August, 2010.

This report results from the design project completed during term where our group developed and built a device that wirelessly managed the load on an electrical circuit. We were engaged in the various steps of converting a concept to functional product. This report provides details of how that task was accomplished.

This course gave us an opportunity to apply technical knowledge, develop team building skills, and cultivate a sense of the economic justification involved in realizing an engineered product.

We would like to thank you for your proposal that was the basis of our project, and for your patience and commitment in supervising us.

Sincerely,

Karl Fort, Danish Qureshi
Tom Nute, Steve Smith

Table of Contents

LIST OF TABLES AND FIGURES

Executive Summary

This report details the development and construction of the Happy Plug device. Happy Plug manages the electric load on a circuit by controlling the number of appliances that are turned on at any one time. Typically a circuit in a home has a 15 amp capacity. If too many appliances are attached to one such circuit the load on the circuit may exceed the current capacity of the circuit causing the breaker to trip. Happy Plugs prevents this inconvenience by automatically turning off non-essential appliances based on a power sharing regime, and using wireless technology. The added advantage is that a home owner averts the cost of rewiring a house and saving thousands of dollars.

The device has six major components:

- Bee radios - which provide wireless capability for the various modules to communicate
- Relays - the manual switches that are controlled by the Bee which in turn controls the appliance plugged into the module
- A Micro-Controller - Adriano processing centre for the device
- A Current Sensor- which detects when a priority device is in use
- Power supplies - which convert high voltage source to 5 volts supply for the micro - controller and Bee Modems
- Software – This controls the operation of the Adriano processor

The system has multiple modules. One functions as a coordinator and contains the Adriano processor, an XBee modem, and a relay, so it also has the same capabilities of switching off an attached appliance. The secondary units contain an XBee and a relay and are able to switch an attached appliance on or off. All secondary units are controlled by the coordinator. One of the secondary units may be a 'priority unit' and the operation of an appliance attached to this module causes all the other modules to switch off their attached appliance. This module incorporates a current sensor for sensing the on state of an attached appliance.

The prototype worked according to specifications, but more refinement and development is necessary to achieve a marketable product. The cost to build this prototype was only $100 compared to the cost of rewiring a house which on average is more than $6,000. Using Happy Plug, therefore, is a very cost effective alternative to rewiring a home if there are issues with breakers tripping due to overloaded circuits.

Introduction

Currently houses are built with several different circuits to run various high powered appliances. Each of these circuits has a dedicated circuit breaker that protects against current overload. When the current or load in the circuit exceeds a certain maximum, typically 15 Amps, the circuit breaker (or circuit switch) will trip or turn off the current automatically to protect wires from overheating and potentially causing a fire.

In houses built, say 50 or more years ago, typical circuit layouts were not designed to accommodate many of the high powered appliances used today. Therefore, the number of different circuits included in a single unit was much less than houses built today. As a result, lights, a heater, a microwave, and a clothes iron may all be powered by the same circuit. If these appliances are used all at once the risk of exceeding the load capacity of that circuit becomes significant.

As a cost effective alternative to rewiring such a house, our group designed a power management device that wirelessly controls the number of appliances connected to a particular circuit at any one time, thus maintaining the load on that circuit below acceptable limits and avoiding the inconvenience of a tripped breaker.

This solution was accomplished in two stages. A basic system consisting of master and slave units was constructed. Once it was working, the design was modified to incorporate added features and autonomy. The slave functioned as a remotely controlled switch that turned an attached appliance on or off, and the master incorporated a control schedule that evaluated and decided which appliances would be turned on or off based on circuit loads and need-to-use criteria.

The report discusses the major components of the device in separate sections; Radios, Relays, Micro Controller, Current Sensor, Power Supplies, and Software; then it address the device as a complete and functional entity.

Discussion

Radios

A major feature of the Happy Plug is that it manages electrical outlets wirelessly. Even though there are many options to achieve wireless communication, XBee Pro, Series 1, radios were used. This choice was based on the fact that XBee Modems are relatively inexpensive, simple to use, low power, and are very popular among hobbyists who generally provide easily accessible information on how to use and program these devices. The XBee Pro, Series 1 suited the functional needs of this project well.

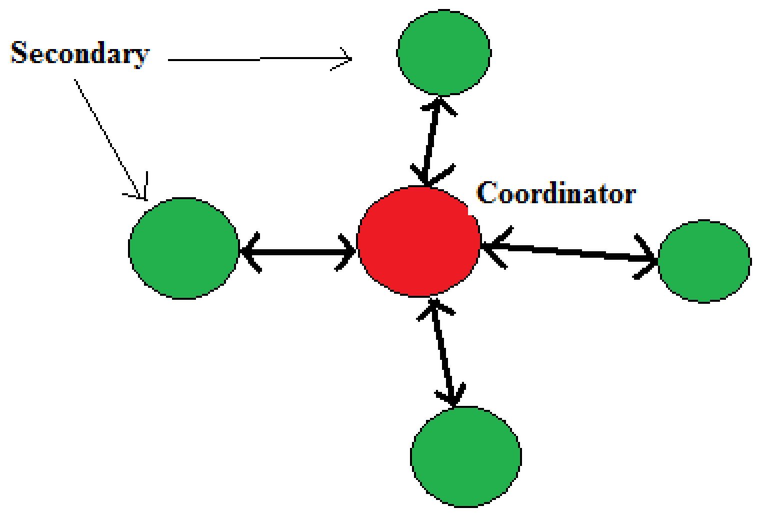


Figure 1: Network topology of project

The design of Happy Plugs required communication between a coordinator device that controlled the schedule of switching times, and secondary devices that were switched in response to the commands from the coordinator (see Figure 1 above). The coordinator device was attached to an Adruino micro-controller, the processing centre for the system. All commands were issued through the XBee radio attached to it and received by other XBee radios attached to the secondary units, one of which was a ‘priority unit’ that was configured to return a signal from the current sensor attached to it. Each secondary unit controlled the appliance plugged into it.

To achieve the necessary functionality each XBee was configured using the X-CTU software running on Windows.   The parameter setting for the XBee modems are outlined in Table 1.

Channel	C	D
Pan ID	E499	E499
Destination Address High	13A200	13A200
Destination Address Low	4049247B	4049248F
16 Bit address	FEFE	FFFE
Serial # high	13A200	13A200
Serial #low	40640C6D	4069248F
Coordinate Enable	0	0
Node ID	Yellow	Red

Table 1: XBee parameter settings using X-CTU software

Figure 3 shows the pin layout of the XBee and Table 1 gives a description of the various pins.

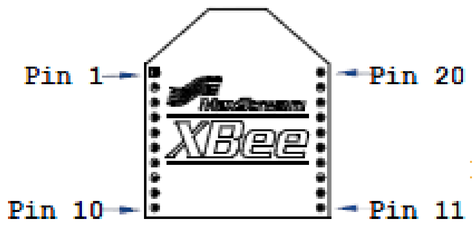
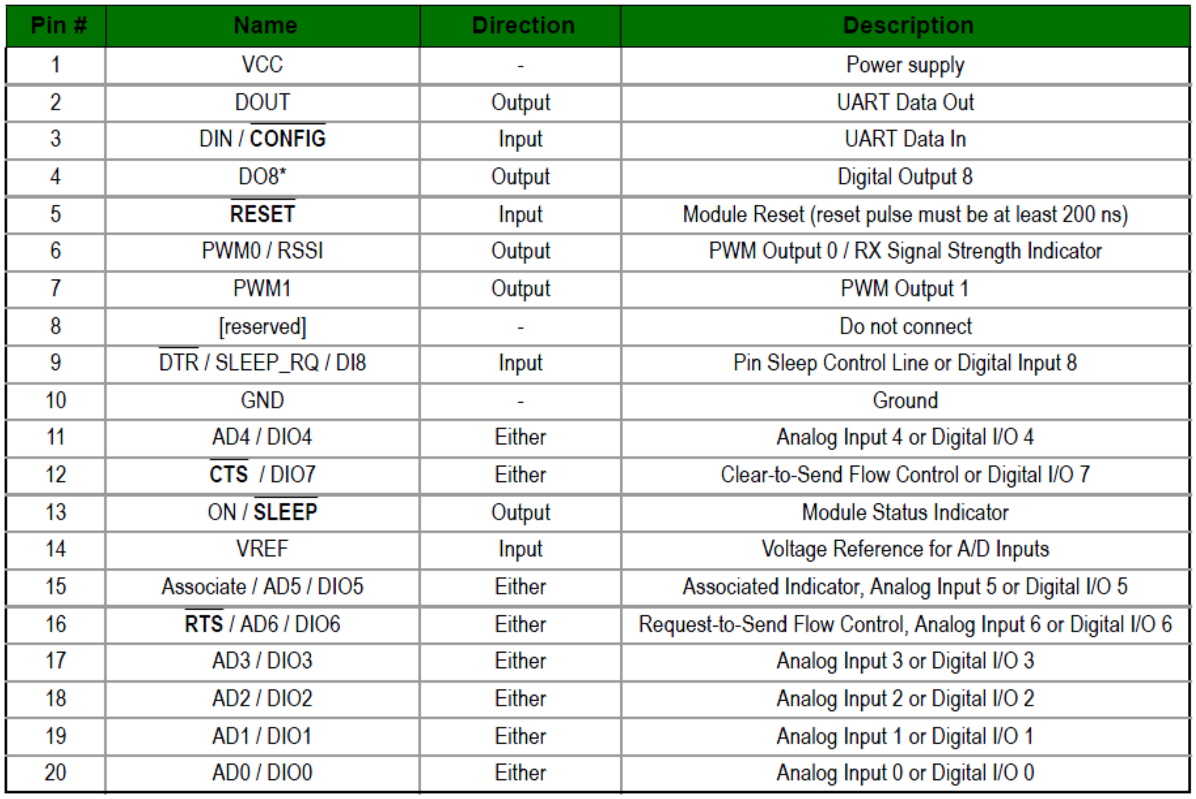


Figure 2:   Pin layout of XBee



Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / $\overline{\textbf{CONFIG}}$	Input	UART Data In
4	DO8*	Output	Digital Output 8
5	$\overline{\textbf{RESET}}$	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	PWM1	Output	PWM Output 1
8	[reserved]	-	Do not connect
9	$\overline{\text{DTR}}$ / SLEEP_RQ / DI8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog Input 4 or Digital I/O 4
12	$\overline{\textbf{CTS}}$ / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / $\overline{\textbf{SLEEP}}$	Output	Module Status Indicator
14	VREF	Input	Voltage Reference for A/D Inputs
15	Associate / AD5 / DIO5	Either	Associated Indicator, Analog Input 5 or Digital I/O 5
16	$\overline{\textbf{RTS}}$ / AD6 / DIO6	Either	Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

Table 2: Pin assignment for XBee Pro

While programmed pins 1 – 10 interface with the computer, pin 11 is used in detecting the input from the current sensor in the 'priority' device independent of the computer, and pin 4 outputs the signal that switches the relay controlled by each secondary unit.

The XBee modems use the 802.14.4 Protocol on a 2.4 GHz broadcast frequency and have a range of 30 metres indoors. In order for the coordinator to distinguish between individual secondary units, each unit is assigned a unique 64 bit ID or address.

Relays

Relays were chosen based primarily on cost. Given that a standard household circuit breaker is rated for 15A and that the purpose of our project is to switch devices that draw almost this amount of current, the relays themselves had to be rated similarly. This meant handling both soft and hard loads (resistive and mixed resistive/inductive) up to 15A. Additionally the relays had to be rated for use on the 120V AC line voltages used in North American household electrical systems. In order to market the product internationally this rating must be increased to 240V AC. For the prototype a relay rated for 30A resistive load at 240V AC from Tyco's (Potter & Brumfield) T9A series was used.

Figure 3: Relay

The T9AS1D12-5 has a minimum coil voltage of 5V and pulls approximately 180 mA at this level from our testing. This makes direct switching by the Arduino or XBee impossible as the outputs of these devices are rated for less than 25mA. The solution to this problem is to use a transistor based relay drive circuit as shown in Figure 4 below.

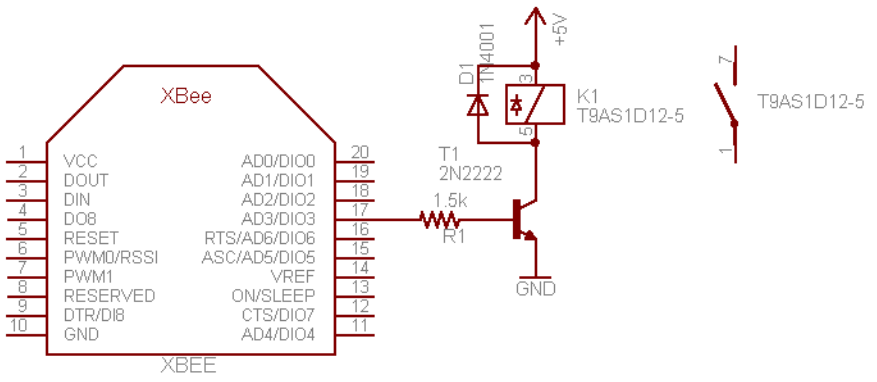


Figure 4: Transistor based relay

Unfortunately, these relays are only rated for 100,000 operations, corresponding to a lifetime of only 2.85 years if used to switch a load on and off in 15 minute cycles. This could be a major issue and is one that must be worked out before a final product design is reached.

Both the switching voltage/current issue and the lifetime issue could be eliminated by the use of solid state relays (SSRs) but prices for these devices are still significantly higher than for mechanical relays at the ratings we require.

Microcontroller

It was clear early on in the project that a microcontroller would be required to get the kind of functionality desired. As the brain of the system the micro-controller is tasked with knowing at all times what other units are present in a system and building a load sharing schedule based on this information. These tasks are not terribly complicated or intensive, so standard, inexpensive, 8-bit microcontrollers were looked at for use in the design. Atmel's 8-bit AVR and Microchip's PIC microcontroller were looked at first as they are the most popular choices for high volume low cost embedded systems. To further simplify development there are platforms built on these chips that allow them to be easily programmed and interfaced with external devices. One option, the "Basic Stamp" is widely used for prototyping as well as hobbyist applications and is based on the PIC microcontroller. Similarly, the "Arduino" development platform is based on Atmel's AVR microcontroller. Both platforms are available in a variety of forms depending on the end-user requirements.

The Arduino platform was chosen primarily due to its low startup cost and extensive support. Due to the popularity and open-source nature of this platform the hardware itself is readily available in a huge number of configurations. For the purpose of this project size and cost were of greatest concern over feature set. The version used, Arduino Pro Mini 328 v11, contains only the microcontroller, external clock, and reset button on a pcb measuring only 18x32mm. To program the microcontroller on a modern computer, all that is needed is a serial-to-usb converter. Other versions of the Arduino have this feature built in but do so with an increase in size and cost. For the project a single usb-to serial-converter board made by FTDI was used to program the Arduino as well as the XBee radio units. For the Arduino platform software development and chip programming is all done through the free Arduino Integrated Development Environment (IDE). Programming is covered in detail in a separate section but is done using a modified version of the standard C language. All of these benefits made the Arduino an excellent choice to get a prototype running quickly and cheaply to demonstrate to potential investors.

Although the Arduino platform is great for rapid inexpensive prototyping it is unsuitable for final production designs. Size and cost are the main issues when looking at using the Arduino in a commercial product. For the final product a custom PCB needs to be designed and the code either translated or rewritten from scratch. Basing the final design on the same chip (Atmel 8-bit AVR) would simplify the transition stage from prototype to production. This would require a considerable amount of resources to build.

Current Sensing Circuit

Requirement

The current sensor circuit was used in the Happy Plug 'Priority unit'. It was required particularly to sense current threshold when an appliance plugged into the Priority unit was turned ON. The output from the circuit would then serve as the input to the XBEE radio that would transmit this information to the micro-processor in the coordinator device.

The task to design the circuit was difficult because of another design constraint. The circuit had to function such that the output would change states when the input current would pass a particular predefined threshold value. This constraint was put into place because the unit was to be used with a microwave or a similar application which requires power all the time. Even when the appliance is not in use the clock will still require power to work. The current drawn by the appliance just to keep the clock running will not cause an output state change. It will however change output state when the appliance turns on and the current passes a particular threshold value.

Current Sensor

The current sensor used was the ACS714 which works on the principle of the Hall-Effect. This particular current sensor was chosen because it fulfilled all the requirements for the circuit. The sensor operates at 5 V normally but can operate at 3.3V which was better suited to our application. An average microwave uses 15A current when running. This current sensor has a range of magnitude up to 20 A. Other advantages for this device included precise, low-offset and linear functionality and low power loss due to very small 1.2mΩ internal resistance.

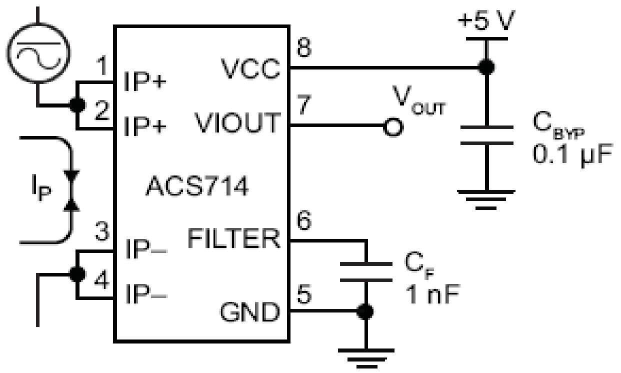


Figure 5: Pin Configuration for ACS 714

Figure 5 shows the basic pin configuration for the current sensor. This figure was studied and used as the basis for the current sensing circuit. Pin 1 and 4 were used to connect the ACS714 IC in series with the power appliance. Figure 6 below shows a schematic for the circuit:

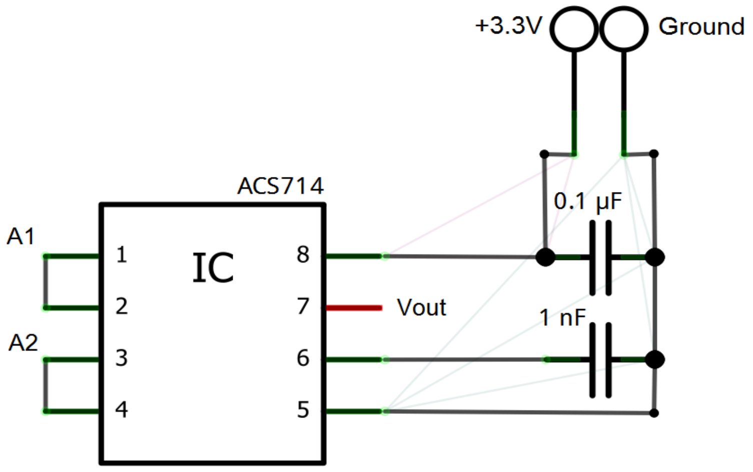


Figure 6: Schematic for Current Sensor

The power appliance was connected in series with points A1 and A2. Output from a power supply set to 3.3V was made to the screw terminals and the output at V_{out} was observed using a multimeter. For experimental purposes a hairdryer with 2-speed settings was connected instead of the microwave. Tabe 3 below shows the results of the observations:

Load	V_{out} AC (mV)
None	2.3
Low	220
High	820

Table 3: AC Output Voltage of Current Sensor

It was observed that the output at V_{out} was AC instead of the expected DC. This was a problem as a DC output was required for the correct functioning of the circuit. Upon troubleshooting it was found that the current sensor was working as it should and it was outputting AC voltage because the input at A1 and A2 was also AC. It was decided that in order to produce a DC output, the AC output from the current sensor needed to be rectified.

Rectifier

The rectifier circuit was needed to convert the AC output from the current sensor to DC. As the application did not require a strict DC rectification a basic half wave rectifier was designed. The figure below shows the schematic for the circuit:

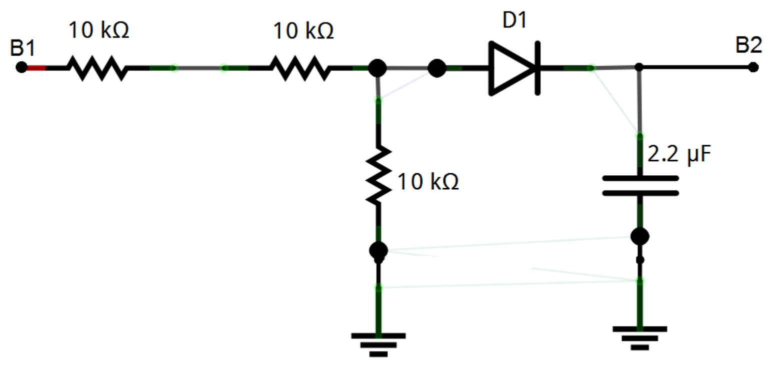


Figure 7: Schematic for Rectifier

The AC voltage is applied at the terminal B1 and the rectified DC output is obtained at the terminal B2.

In order to test the circuit, the output terminal V_{out} from the current sensor was connected to the terminal B1 of the rectifier circuit. The hairdryer was plugged in and the output at terminal B2 was observed using a multimeter by toggling the hairdryer between OFF, LOW and HIGH settings. The following observations were recorded in Table 4 below:

Load	**V_{out} DC (V)**
None	1.830
Low	1.999
High	2.660

Table 4: DC Output Voltage of Rectifier

The results obtained showed that increasing the speed or load caused an increase in the DC voltage output of the circuit. The circuit was thus outputting a linear increase in voltage with increasing input current. The next step was to design a threshold detector circuit which would change states when the rectified voltage passes a reference voltage.

Threshold Detector Circuit

A Threshold detector circuit is a simple voltage switch. It compares two voltages and switches the output to indicate which voltage is larger.

A common device used to perform this function is the comparator which is equivalent to an operational amplifier with two NPN transistors added to the output of the amplifier. The output is switched ON or OFF depending on the relative voltages at the inverting and non-inverting inputs of the comparator. LM393 is a common dual comparator chip used in a number of applications. It provides high precision and has a wide voltage range which made it suitable for our application and was chosen as the comparator chip. The pin connections for LM393 are given in Figure 8 below.

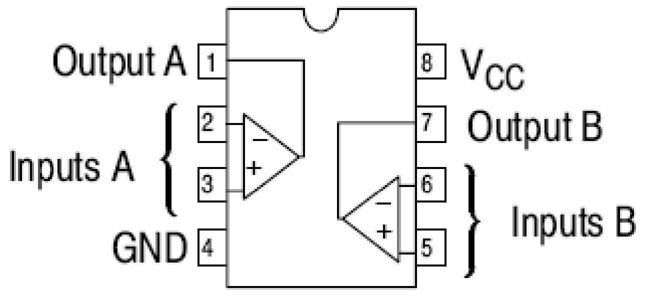


Figure 8: Pin Layouts for the LM 393 Comparator

As our application required the use of just one comparator, pins 5 to 7 were left unconnected. The following diagram shows the schematic of the circuit that was put together for test purposes before the actual circuit was implemented.

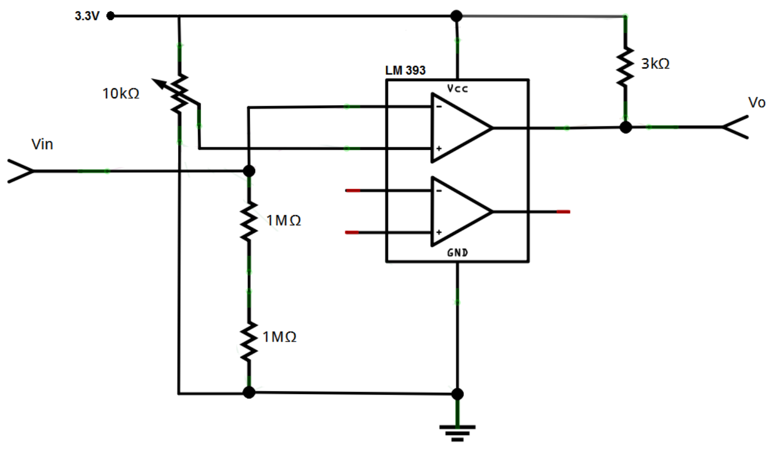


Figure 9: Schematic for Threshold Detector Circuit

The potentiometer was used to set a reference voltage of 1.85V at the non-inverting input. A potentiometer was connected at V_o and a separate power supply was then connected to the Vin terminal with the initial voltage set to 0V. The voltage was steadily increased and it was observed that the output V_o changed from high 3.3V to low 0V when voltage at V_{in} crossed the 1.85V marker. This showed that the comparator was performing as expected and could be used for our application. The graph below shows the behavior for the comparator inputs and output.

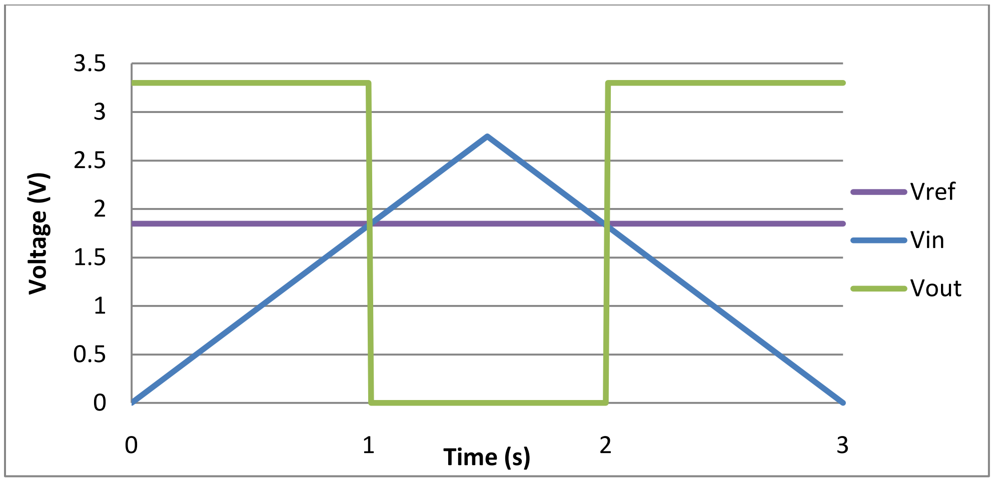


Figure 12 Graph showing change in Voltage vs. Time

Combining

Once all three parts were put together and tested separately, they were combined on one breadboard. Care was taken to transfer every component from each separate part and the connections were repeatedly checked for accuracy. The following diagram shows the complete schematic for the current sensing circuit:

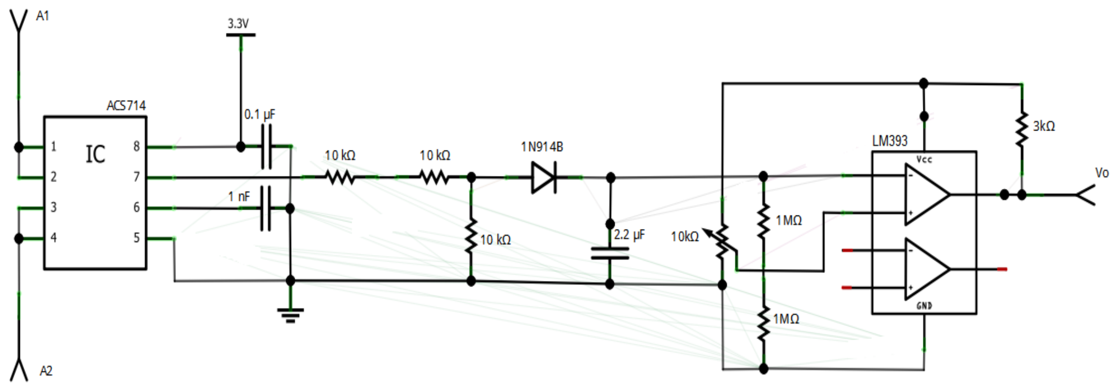


Figure 10: Schematic for the Complete Current Sensing Circuit

Terminals A1 and A2 were connected in series with the hairdryer. The potentiometer was set such that the non-inverting input to the comparator was 2.1V. The hairdryer was set to different speed settings. The voltmeter was used to take the voltage reading for both the inverting input of the comparator and ouput voltage are Tigure 5.

Load	$V_{in\_inverting}$ (V)	Vo (V)
None	1.83	3.3
Low	1.99	3.3
High	2.66	0

Figure 11: DC Input and Output Voltages of Current Sensing Circuit

It can be observed from the table that the circuit outputs 3.3V when there the hairdryer is OFF or at LOW speed. However when the hairdryer is turned to HIGH speed, the circuit outputs 0V. The output of this circuit V_o can thus be used as an input for pin 4 of XBEE. When the appliance connected in series with terminals A1 and A2 is consuming low power there would not be any change to the pin 4 of the XBEE. However, when the appliance connected will start consuming high power, the input to pin 4 will change from high to low.

Power Supplies

To provide power for the units AC-DC converters were needed. From measurements taken the relays draw approximately 180mA at 5V DC, the Arduino 80mA, and the XBees 60mA for a combined total of 320mA. To maintain acceptable efficiency and minimal size for the units it was decided that the best option was to use off-the-shelf solid-state switching power supplies. Re-purposing discarded cell phone 5V power supplies yielded appropriate parts for all three demonstration units.

Logic Level Converter

The logic level converter is necessary to enable communication between the microcontroller and XBee radio components due to the different signal voltages these component uses. The Arduino uses logic signals of 5V and 0V while the XBees use 3.3V and 0V, and will suffer permanent damage if 5V is applied to their pins. The ideal

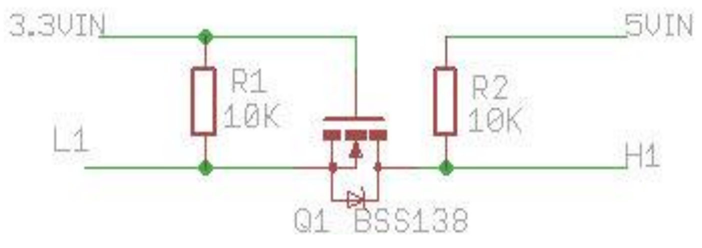


Figure 12: Logic level converter schematics

solution would be to use 3.3V throughout the system but due to time constraints it was better to simply introduce a logic level converter circuit between the XBee and Arduino. Level conversion is a common problem in digital systems as more and more chips switch to using 3.3V so there are many documented methods available. The use of protective diodes is one of the most common methods but it depends on the correct polarity being used according to signal direction and protection requirements. With this method, if wires are plugged in incorrectly (a very real possibility when dealing with circuits on a breadboard) the XBee radios would be rendered useless. The circuit shown in Figure 12 acts as a bi-directional level converter, seamlessly allowing the Arduino's 5V serial transmit and receive pins to be connected to the XBee's corresponding 3.3V receive and transmit pins.

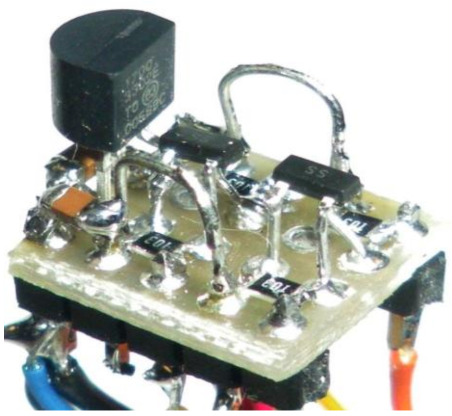


Figure 13: Bidirectional logic level converter

The circuit works using a pull-up resistor on each side of a MOSFET. The MOSFET's inherent body diode prevents the higher 5V voltage from appearing on the 3.3V side while the MOSFET itself allows current to be sunk when either side is pulled low, bringing the opposite side low. For our purposes a simple circuit consisting of two of these modules was created on a piece of veroboard using surface mount components to reduce size and cost. The circuit worked exactly as planned, allowing seamless communication between the XBee and Arduino while preventing costly hardware failure.

Software

When approaching the code, the main objective was to create a system that could periodically turn devices on and off, as well as receive aperiodic messages from a priority device. Once a satisfactory system was developed capable of these two processes, the system needed to be extended to perform the communication via external components. This software section will detail the original on-paper design as well as its changes and implementations on the Arduino microcontroller. Afterward, the integration of XBee radios and how the system uses them will be detailed. Third, the alteration of the system to account for multiple circuits will be explored. Finally a summary of the final software system is shown.

Basic Operation – Theoretical Design

The following describes the system in an ideal setting. Afterwards, the restrictions and limitations of the Arduino microcontroller are considered and the changes to the design are discussed.

When considering the periodic priority messages, the system could implement one of two solutions. Either the system could continually poll for a message or the message receipt could generate an interrupt. Similarly, handling the periodic device switching could be accomplished through either time delays (sleep) or through the use of a timer interrupt. Any combination of these solutions should in theory accomplish the desired results. In the end, the decision was to use interrupts to implement both tasks. This decision was made based on a couple of factors. Firstly, by using interrupts for both, the system can be put into a sleep mode while waiting. This would drastically reduce the power consumption of the system which is a major factor to consider in real-time embedded systems. Secondly, if polling were used for the priority messages, implementing a time delay for the periodic switching would have been more difficult, so an interrupt system would have been required for device switching. For readability, maintainability, and convenience, it was decided to keep both operations using similar ideas, so interrupts were settled on.

With the implementation of the operations decided on, a basic system design was drawn up. As detailed in Figure 14, the system consists of four basic states plus the initialization state. All red transitions indicate transitions caused by interrupts. The "Init" state would consist of the setup of all the interrupt service routines (ISRs), the timer, and powering on the starting device. The system would spend most of its time in the "Sleep" state where is simply waits for interrupts. The "Swap Devices" state is simply the periodic timer's ISR. In this ISR, the signals to power off a device and power on a device would be sent and then control would be returned to the "Sleep" state. The "Priority" state represents the ISR called from the priority message interrupt. This ISR would first power off a device and then put the system in a disabled state. Inside the disabled state, any timer interrupts would be ignored. Once the priority message is received indicating the device is off, the ISR would return the system to the "Sleep" state.

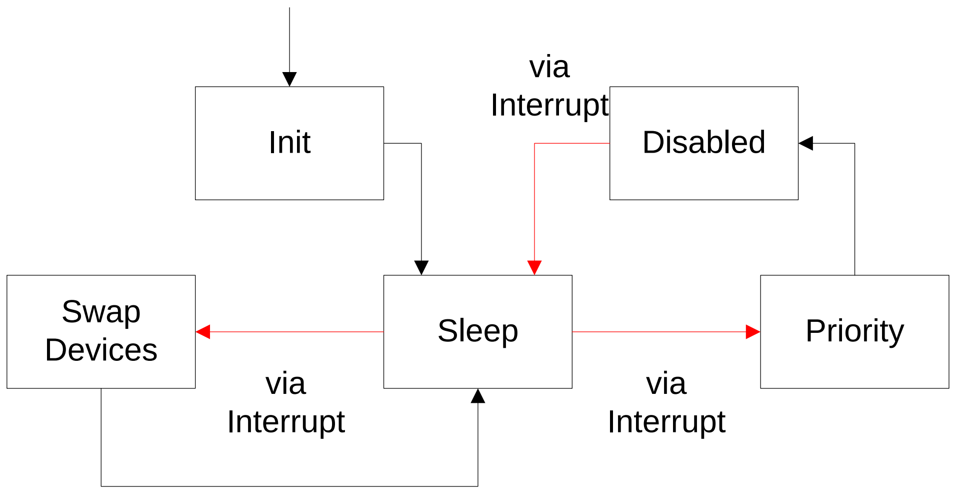


Figure 14: Basic System Design

Basic Operation – Arduino Implementation

The original design is what was used when first attempting to implement the system on the Arduino microcontroller. The first problem encountered was the difficulty in accessing and extending the Serial Interrupt Service Routine on the Arduino. This extension would be necessary because the Arduino, like most embedded processors, already has a built-in ISR to support the UART (Universal asynchronous receiver/transmitter). To implement an interrupt driven system for the priority messages, the Serial ISR would need to be altered to include a check to see whether the message is a priority signal or not. With the inability to alter the Serial ISR, the design was altered to poll for messages from priority devices.

The interrupt system for the periodic switching of automatic devices was much easier to develop. The operation was implemented utilizing a user created date and time library obtained through the Arduino community. The "TimeAlarms" library (which is a companion to the "Time" library) allows for the creation of periodic or one time alarms. It allows you to specify a function to run when the Alarm goes off. This library essentially takes care of all the interrupt setup work and coding, allowing for easily implemented timers.

Figure 15 shows the updated system design. The "Priority" stage from the earlier design has been removed since there is now no need for an interrupt service routine. The actions performed in that state were simply moved into the transition between "Ready" and "Disabled". The polling for messages is shown has diamond decision branches. The "Switch Devices" state and transitions would be exactly the same.

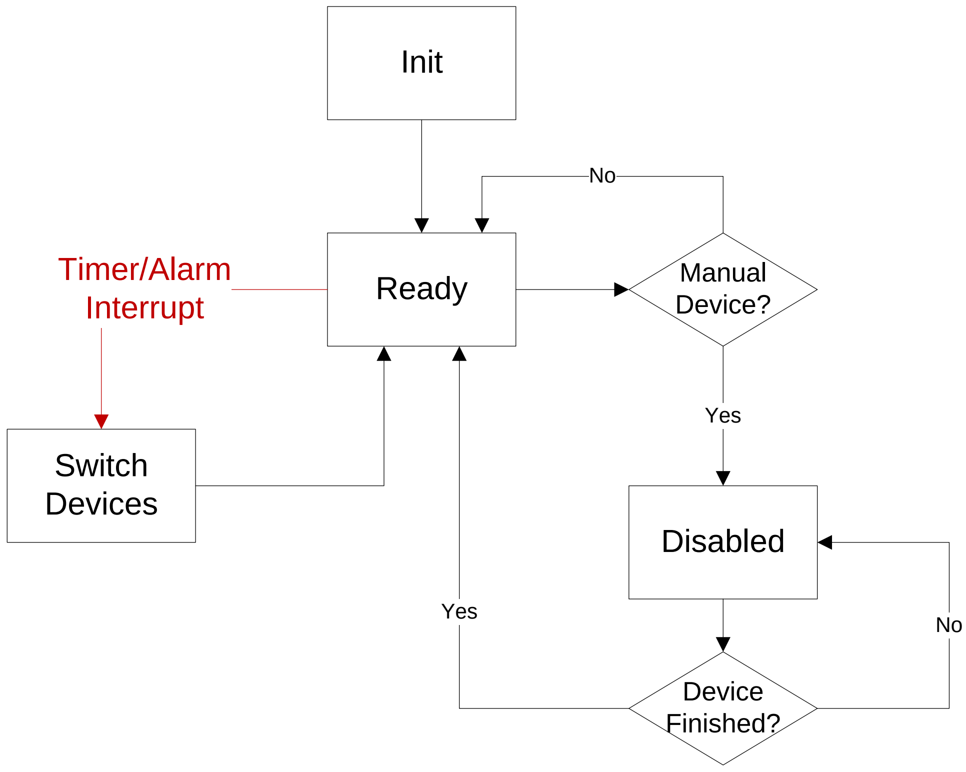


Figure 15: Basic System Design implemented on Arduino

Integrating the XBee Radios - Networking

To incorporate wireless communication, the system needed to be connected to XBee radios (See the XBee section of the report for more information regarding the radios themselves). The XBee radios operate in either AT/Command mode or in API mode. In both operational modes, the XBee and Arduino communicate via the serial channel. In AT mode, the Arduino sends commands comprised of ASCII encoded characters (ie. “ATND” or “ATIS”). The XBee responds with strings of characters in specified formats, terminated by multiple carriage returns. This is the simplest way to communicate with the radios so the first step was to use this mode to implement the part of the Initialization state that required the system to establish the network.

Sending the node discovery command (ATND) from the Arduino to the local XBee (local refers to the radio physically connected to the Arduino) causes the radio to broadcast to all other XBees, asking for information. Each XBee receiving this request responds with their name, unique address, network ID, and signal strength. The local XBee then retransmits this information over the Serial port to the Arduino. The system reads a number of bytes of the serial input corresponding to the length of the information for a single device. The system creates a struct to store the information and then repeats the process with the next device's information, creating a list of data structures representing the radios on the network. Finishing this, the system enters the Ready state.

Integrating the XBee Radios – Signalling

Once the network was established and the Arduino had all the information necessary, the system needed a way to actually send messages to remote radios. Remote commands, commands that are passed on from the local radio to a remote one, cannot be executed using the simple AT mode of operation. Remote commands require the use of the API operation mode which was not something originally anticipated when implementing the node discovery operation using AT mode.

API mode requires the creation of a message "frame". This frame is simply an array of bytes with certain indexes having pre-defined uses. Figure 3 is taken from the XBee user manual. It lays out how a Remote AT Command Request should be formulated.

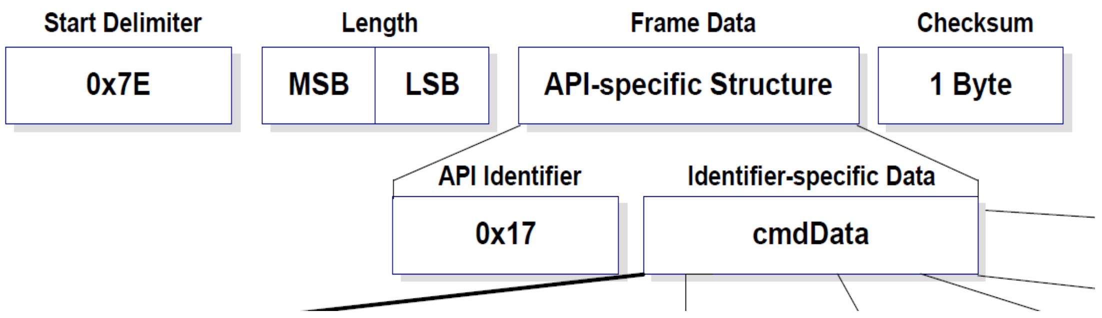


Figure 16: XBee AT Command Request Frame layout

It turned out that the creation and sending of these frames was relatively straight-forward. At this point, the only use for the remote command request was to execute a digital output command. The Arduino simply needed to specify which pin, ranging from 0-4, and whether the change should be too high or low. Sending the frame merely consisted of writing the byte array to the serial port, connected to the local XBee. With a simple function to create and send these frames, the Arduino was able to remotely signal other XBees to set a pin, which we had connected to a relay, high or low, essentially allowing the Arduino to turn on and off an outlet remotely.

Integrating the XBee Radios – Receiving Priority Signals

The XBee connected to the priority device had been setup to automatically send out a signal to the coordinator XBee (the local XBee connected to the Arduino) whenever one of its digital input pins changed. The signal it sent was a standard sample signal. These samples that the XBees are able to send consist of a value for all pins configured as inputs along with some metadata. It was expected that the Arduino would simply receive a sample via the local XBee and read the specific byte related to digital pin 3 (the pin on the radios we had connected to the priority device). In practice however, this was much more difficult.

For whatever reason, the data the Arduino was receiving never matched up to the template provided in the user manual. After much frustration and work, it was discovered that a library for the XBee API was available through the Arduino community. This library provided a simple interface for extracting data from sample signals. With the use of the library, it became trivial for the Arduino to know when the priority device was turned on and turned off.

Advanced Operation – Multiple Circuits

Late in the project, it was decided that the system should be able to coordinate multiple circuits with a single controller system. Basically, the idea was to allow a single automatic device to be on per circuit. Each circuit could have a manual device, which while on would disable the automatic devices on the same circuit only. This alteration to the system specifications required changes, some trivial and some more difficult ones.

The first thing to change was the "Switch Devices" state in the earlier design. Originally, the system simply turned off the device currently on and then powered on the device next in the list. To accommodate multiple circuits, this on/off switch simply needs to be executed multiple times, once for each circuit. For each circuit, the routine must also check whether or not the priority device associated with that circuit is on or off. This simple routine is given in pseudo code below.

```
For each circuit
    If manual circuit is not on
        Turn off device currently on for circuit
        Turn on next device for circuit
```

Initially this process seemed perfectly fine. With some initial testing however, one flaw was discovered. Depending on when the priority device was turned on and for how long, the automatic device that was interrupted may end up receiving very little on time.

Because all the circuits were serviced by a single alarm, if the priority device came on shortly after an alarm and then went off shortly before the next one, the automatic device on that circuit would get swapped only having received a short amount of time. The main consideration when thinking about whether this was a problem that needed fixing was the fact that the current implementation of the system had no way of getting feedback from automatic devices, such as temperature (although this is certainly a possible future enhancement). The standard use case for the system uses heaters for automatic devices. Not giving each heater the right amount of time would likely result in temperature issues. Without a feedback, it was decided that it was important each device receive the specified amount of time.

To solve this issue there were two possible solutions. After returning from being interrupted by a priority device, the disabled automatic device would have its time on reset to the original amount. The other choice was the give the automatic device the time remaining of its total when interrupted. The first solution is good in situations where the priority device is on for an amount of time that is relatively large compared to the time an automatic device gets. However, if priority devices are expected to be on for fairly short periods of time then this poses a problem. If the automatic device was on for the majority of its allotted time only to receive another full allotment after being interrupted for a short time, you may run into over-use issues (over-heating in the case of heaters).

The best solution for the system was to give the automatic device the amount of time it had remaining when interrupted. So if the device was interrupted 10 minutes into a 30 minute total, it would receive 20 minutes after interruption. While a good solution to device starvation, this idea also introduced another potential problem of its own. Devices could no longer be swapped all at the same time as their durations could end at differing times depending on priority device use. The initial reaction to this was to have each circuit have its own alarm. After some analyzing though, this idea adds far more overhead than is necessary. At system start-up and for some time after, all device-switching will be performed at the same time. Therefore, a single alarm can be used up until a priority device is used. Afterwards, alarms need to be created on the fly. This system sounds complicated at first but it can easily be described using the pseudo code below.

```
Function mainLoop
    ...
    For each circuit
        set first device's expectedExpiry to time interval + now()
    create Alarm for time interval given

Function priorityDeviceON
    power off device on this circuit
```

```
        remember how much time the device had remaining

    Function priorityDeviceOFF
        power on device on this circuit
        retrieve how much time the device had remaining
        set expectedExpire to remaining + now()
        create Alarm for time remaining for device

    Function AlarmExpiry
        For each circuit
            If manual circuit is not on
                If current device's expectedExpiry before now()
                    Turn off device currently on for circuit
                    Turn on next device for circuit
                    Set device's expectedExpiry to interval + now()
        End For
        Create Alarm for time interval only if ANY device was swapped
```

The first thing that needed to be added was the variable, "expectedExpiry", one for each circuit. This is the time that the device will need to be swapped. It is an absolute time as opposed to an interval and is created initially by adding the time interval to the current time returned by "now()" function. Initially all the circuits will be serviced by a single Alarm that is scheduled to go off after the time interval. The circuits will continue to be serviced by a single alarm until one is interrupted by a priority device. After the interruption ends, a new Alarm is created to go off after the time remaining for the automatic device on the given circuit. In this way, the additional overhead of multiple alarms is only added on an as-needed basis; a separate alarm is only created when a circuit has its first interruption via a priority device. At maximum, there could be one alarm per circuit.

Recommendations and Future Enhancements

The system implementation uses three distinct methods for communicating with the XBee radios; AT Command mode, API mode (manually), and API mode with the Arduino-XBee library. The first thing to change in future revisions of the system would be to refactor all operations to use the library calls for API mode. This will increase the readability of the code, both through consistency and the simplicity of library calls, while also improving maintainability.

A possible future enhancement would be a feedback from automatic devices indicating things such as temperature. Though primarily a hardware change, the software's scheduling would have to change. In the current implementation, all devices are scheduled in a basic queue. The only major alteration would be to allow for a signal, indicating a temperature threshold for example, to move a given device to the head of the queue, possibly interrupting the device currently on. The reading of the signal could be done during the polling for priority signals. The system could just poll for any message and do different handling based on the type.

Final Product

Having a prototype that demonstrates how the final product will look and operate was an important part of the project. The initial concept of the project was to have multiple units that would plug directly into standard outlets and that the consumer would plug their device into the bottom of the unit. An initial sketch was drawn up and can be seen in the figure on the right.

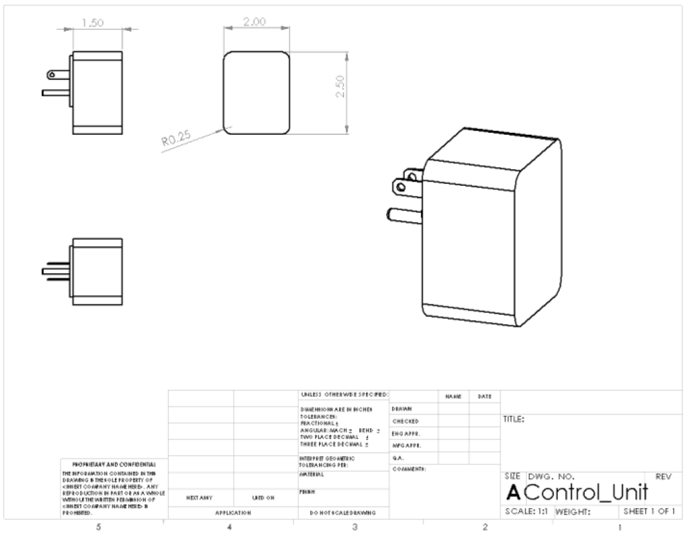


Figure 17: Schematic of enclosure design

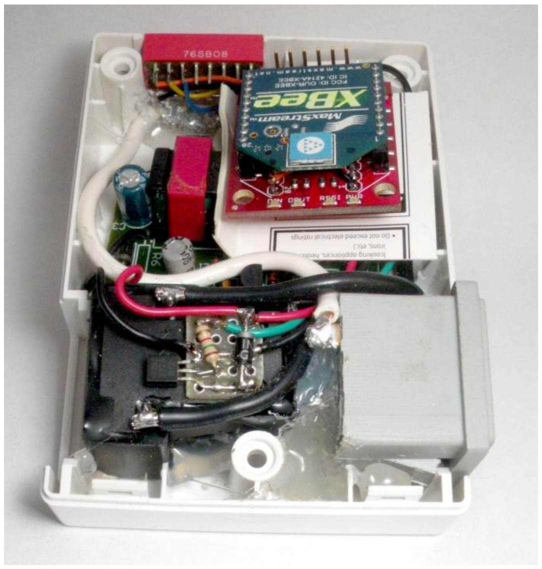


Figure 18: Completed prototype

For the demonstration prototype it was impractical to have a custom built unit manufactured to specifications and so alternatives had to be examined. Looking at devices on the market that are of a similar form, (small box, male connector on one side, female connector on another) it was determined that the best solution would be to re-purpose the enclosures from one such device. Devices such as single outlet surge protectors, multi-tap outlets, X-10 units, and digital/mechanical outlet timers were researched to find an enclosure that would suit our requirements. The use of project boxes from electrical suppliers as well as surface mount outlet boxes was also looked at. For the demonstration prototype the GE 15271 mechanical outlet timer was selected. Available locally for $15.00 a unit this was one of the most economical options and required the least amount of modification to adapt the enclosure for the project. Removing one of the unit’s two outlets as well as the mechanical switch assembly left enough room to house the power supply, relay, microcontroller, logic level converter, and Xbee needed for the control unit.

Despite such challenges, however, the project was a success. The system successfully switched secondary devices on a time schedule when the ‘priority’ device was off and switched off all other devices when the ‘priority device was switched on.   The system was not tested on a circuit that was likely to become overloaded and trip a breaker but it was able to turn off all appliance within a forty second time limit which is a design constraint based on the time it takes to trip a breaker at 30 Amps.

Market Analysis

The cost to build this prototype was $200. With streamline mass production and economy of scale the end product could sell for less than $50. Similar products exist, XBee Smart Plug for example, at a cost of $100. It has very similar features and uses Digi’s technology.

A survey of Electricians in Victoria indicated that rewiring a house to add circuits, for an 80 year old home, on average costs more than $6,000. Victoria has a large number of old houses and rewiring jobs are a common occurrence.

Conclusion

The inconvenience of overloaded household circuits is an issue most people can relate to. Up until now the only solution was the costly and often impractical upgrade of existing wiring by a qualified electrician. This project is aimed at providing a marketable product that solves the problem in the most inexpensive, flexible, and consumer friendly manner possible. By using plug-and-play units that perform unique functions there is almost zero programming required by the end user other than plugging in their devices and setting the circuit number switch. The use of Digi's XBee radio units, based on the IEEE 802.15 standard, allows the devices to form their own wireless network with a range that should be large enough for any home.

By using three different units (Coordinator, Secondary, and Priority Sensor) hardware costs are minimized as each unit only contains the hardware required to do its job. The coordinator contains an XBee radio, microcontroller, relay, and power supply that allow it to manage all of the other devices in the system. Its job is to keep a schedule based on the other units in the system as well as turn on and off its own relay. The secondary devices simply turn on and off their relay according to instructions from the coordinator. The priority device is for the situation when a consumer wants to use an appliance at the same time as another less important device is running. Using a Hall Effect current sensor to detect the operation of whatever is plugged into it the priority sensor simply sends out a signal for other units on the same circuit to turn off. The priority sensor does not contain a relay and thus is incapable of switching local loads.

The prototype presented in this report serves as an excellent demonstration that the concept works and is a marketable product. At this stage the design and functionality of the product has been worked out in detail and is at the point where the design must be optimized in terms of size and cost in order to bring it to market. Tasks remaining include designing a custom PCB for each of the three unit types that integrates all of the components that were previously on discrete PCBs. Simultaneously, an enclosure must be created that will be able to fit the components and be attractive to consumers. The testing and certification of the device is another issue that will need to be addressed. Overall this is a very solid product with excellent marketing potential thanks to its unique design solutions.

References

[1] Philips, Appl. Note 97055, pp.10-16.

[2] Digi International, "XBee /XBee-PRO RF Modules,"Product Manual, Sept. 2009.

[3] ATmel, "ATmega48A/48PA/88A/88PA/168A/168PA/328/328P," Product Summary, Apr. 2010.

[4] ATmel, "8-bit AVR Microcontroller with 4/8/16K Bytes In-System Programmable Flash," ATmega 328 Datasheet, Oct. 2009.

[5] Tyco Electronics, "DC Coil 30 Amp PC Board or Panel Mount Relay," T9A Series Datasheet, Mar. 2003.

[6] Allegro Microsystems Inc, "Automotive Grade, Fully Integrated, Hall Effect-Based Linear Current Sensor IC with 2.1 kVRMS Voltage Isolation and a Low-Resistance Current Conductor," ACS714 Datasheet, 2010.

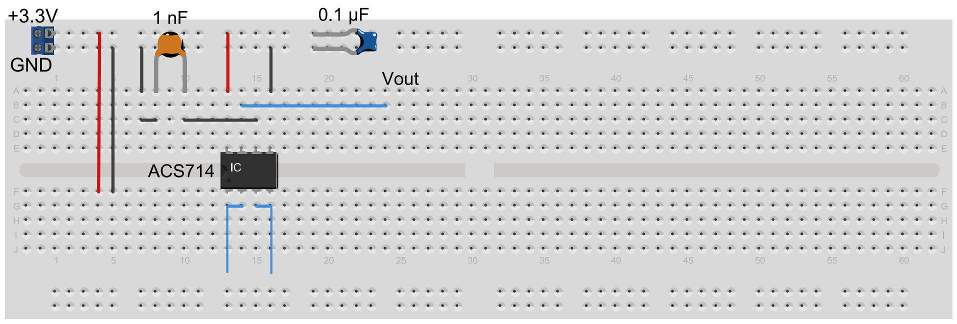
[7] Sharp, "IT(rms)=16A, Non-Zero Cross type SIP 4pin Triac output SSR," S116S01 Series Datasheet, Apr. 2004.

[8] Allegro Microsystems Inc, “Automotive Grade, Fully Integrated, Hall Effect-Based Linear Current Sensor IC with 2.1 kVRMS Voltage Isolation and a Low-Resistance Current Conductor,” ACS714 Datasheet, 2010.

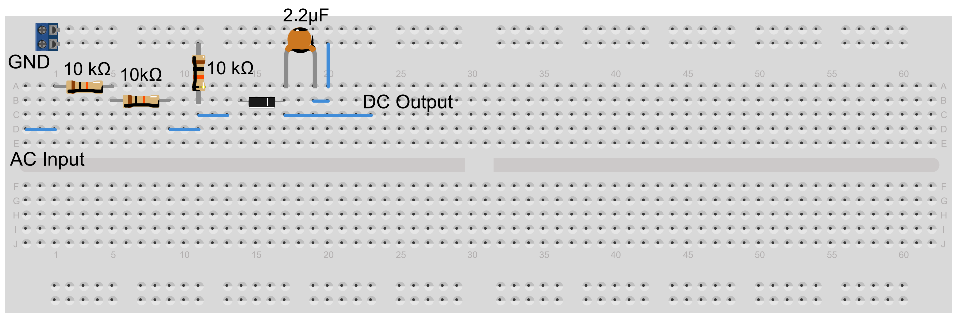
[9] National Semiconductor Corporation, “LM193/LM293/LM393/LM2903 Low Power Low Offset Voltage Dual Comparators,” LM393 Datasheet, August 2002.

[10] Fairchild Semiconductor Corporation, 1N/FDLL 914/A/B / 916/A/B / 4148 / 4448 Small Signal Diode,” 1N914B Datasheet, January 2007.

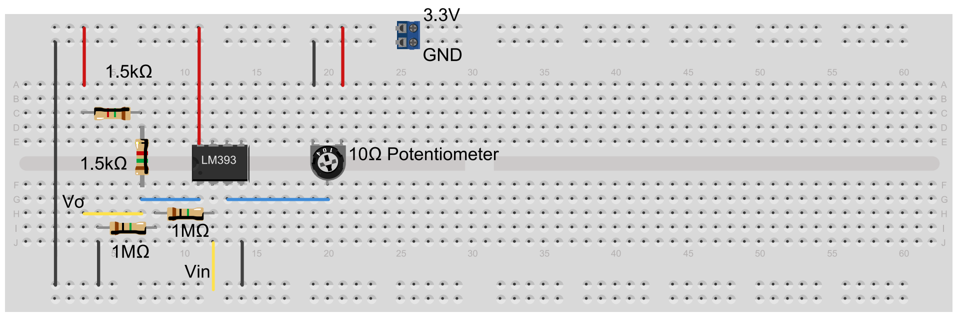
Appendix A



Breadboard View for Current Sensor



Breadboard View of Rectifier



Breadboard view for Threshold Detector circuit

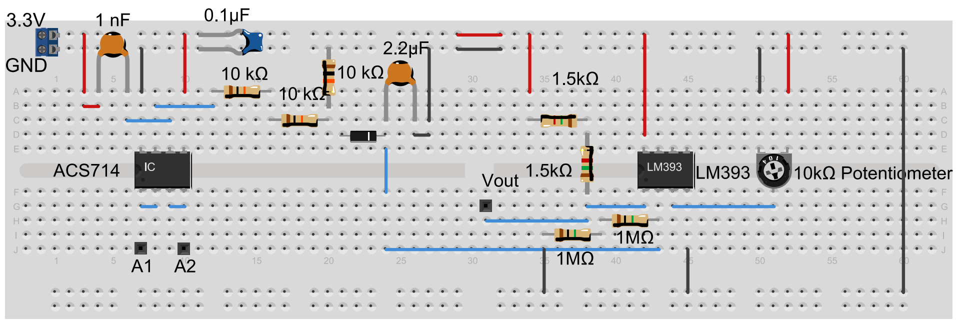


Figure 19: Breadboard View of the Complete Current Sensing Circuit

Bill of Materials

Coordinator

Name	Part #	Manufacturer	Supplier	Supplier Part #	Price	#	Total	Notes
ATMEGA328 Microcontroller	ATMEGA328-AUR	Atmel	Digi-Key	ATMEGA328-AUR-ND	2.60	1	2.60	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=ATMEGA328-AUR-ND
Relay SPST-NO Sealed - 30A	T9AS1D12-5	Potter & Brumfield	Digi-Key	PB1014-ND	3.02	1	3.02	http://search.digikey.com/scripts/DkSearch/dksus.dll?vendor=0&keywords=T9AS1D12-5
XBee 1mW Chip Antenna	XB24-ACI-001	Digi/Maxstream	Digi-Key	XB24-ACI-001-ND	20.91	1	20.91	http://www.sparkfun.com/commerce/product_info.php?products_id=8664
CRYSTAL 20.000 MHZ 20PF SMD	ECS-200-20-5G3XDS-TR	ECS Inc.	Digi-Key	XC1505TR-ND	0.59	1	0.59	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=XC1505TR-ND
Resistor 1.8K 1/16W 5% SMD	CRCW04021K80JNED	Vishay/Dale	Digi-Key	541-1.8KJTR-ND	0.005	1	0.01	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=541-1.8KJTR-ND
Resistor 10K 1/16W 5% SMD	CRCW040210K0JNED	Vishay/Dale	Digi-Key	541-10KJTR-ND	0.005	1	0.01	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=541-10KJTR-ND
2N2222 SMD Transistor	2N2222AUA	TT Electronics	Digi-Key	2N2222AUA-ND	0.13	1	0.13	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=2N2222AUA-ND
3.3V SMD Voltage Regulator	MCP1700T-3302E/TT	Microchip	Digi-Key	MCP1700T3302ETTTR-ND	0.31	1	0.31	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=MCP1700T3302ETTTR-ND
Outlet NEMA 5-15R	738W-X2/02	Qualtek	Digi-Key	738W-X2/02-ND	0.44	1	0.44	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=738W-X2/02-ND
CAP CER 1.0UF 6.3V 10%	GRM155R60J105KE19D	Murata	Digi-Key	490-1320-2-ND	0.007	1	0.01	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=490-1320-2-ND
Power Supply							0.00	
PCB (estimate based on 2"x2")	-	Golden Phoenix	-	-	0.65	1	0.65	http://www.goldphoenixpcb.biz/special_price.php
Enclosure (estimate)					2.00	1	2.00	
Total							**30.67**	

Relay Unit

Name	Part #	Manufacturer	Supplier	Supplier Part #	Price	Quantity	Total	Notes
Relay SPST-NO Sealed - 30A	T9AS1D12-5	Potter & Brumfield	Digi-Key	PB1014-ND	3.02	1	3.02	http://search.digikey.com/scripts/DkSearch/dksus.dll?vendor=0&keywords=T9AS1D12-5
XBee 1mW Chip Antenna	XB24-ACI-001	Digi/Maxstream	Digi-Key	XB24-ACI-001-ND	20.91	1	20.91	http://www.sparkfun.com/commerce/product_info.php?products_id=8664
Resistor 1.8K 1/16W 5% SMD	CRCW04021K80JNED	Vishay/Dale	Digi-Key	541-1.8KJTR-ND	0.005	1	0.01	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=541-1.8KJTR-ND
2N2222 SMD Transistor	2N2222AUA	TT Electronics	Digi-Key	2N2222AUA-ND	0.13	1	0.13	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=2N2222AUA-ND
3.3V SMD Voltage Regulator	MCP1700T-3302E/TT	Microchip	Digi-Key	MCP1700T3302ETTTR-ND	0.31	1	0.31	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=MCP1700T3302ETTTR-ND
Outlet NEMA 5-15R	738W-X2/02	Qualtek	Digi-Key	738W-X2/02-ND	0.44	1	0.44	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=738W-X2/02-ND
Power Supply							0.00	
PCB							0.00	
Enclosure							0.00	
Total							**24.82**	

Sensor Unit

Name	Part #	Manufacturer	Supplier	Supplier Part #	Price	Quantity	Total	Notes
XBee 1mW Chip Antenna	XB24-ACI-001	Digi/Maxstream	Digi-Key	XB24-ACI-001-ND	20.91	1	20.91	http://www.sparkfun.com/commerce/product_info.php?products_id=8664
Current Sensor	ACS712ELCTR-20A-T	Allegro Microsystems	Digi-Key	620-1190-2-ND	1.82	1	1.82	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=620-1190-2-ND
3.3V SMD Voltage Regulator	MCP1700T-3302E/TT	Microchip Technology	Digi-Key	MCP1700T3302ETTTR-ND	0.31	1	0.31	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=MCP1700T3302ETTTR-ND
Outlet NEMA 5-15R	738W-X2/02	Qualtek	Digi-Key	738W-X2/02-ND	0.44	1	0.44	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=738W-X2/02-ND
Comparator	AP331AWRG-7	Diodes Inc	Digi-Key	AP331AWRGDITR-ND	0.13	1	0.13	http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=AP331AWRGDITR-ND
							0.00	
PCB							0.00	
Enclosure							0.00	
Total							**23.61**	

Appendix B

Control Program

```
struct TYPE_device{

int isLocal;

int circuit_id;

char address_low[8];

uint8_t address_low_value[4];

char address_high[8];

uint8_t address_high_value[4];

char source[4];

uint8_t source_value[2];

char signalStrength[2];

uint8_t signalStrength_value;

char name[4];

struct TYPE_device* prev;

};


struct queue_list_t{

  int size;

  struct TYPE_device* head;

  struct TYPE_device* tail;

  queue_list_t* next;

};
```

```
/*
 * Copyright 2010 Thomas Nute.  All rights reserved.
 *
 * This software makes use of XBee-Arduino (Copyright 2009 Andrew
Rapp) under the GNU General Public License.
 *
 *
 */

#include <LiquidCrystal.h>
#include <String.h>
#include "device.h"
#include <Time.h>
#include <TimeAlarms.h>
#include <XBee.h>


#define MAX_CIRCUITS           8
#define MAX_DEVICES            8

#define TIME_READREPLY_TIMEOUT  9000
#define TIME_INTERVAL        30     //Amount of time a device
gets in SECONDS
#define TIME_DELAY_ONOFF     1500  //Delay between powerOFF and
powerON in ISR.. in MILLISECONDS
```

```
#define POWER_OFF            0x04

#define POWER_ON             0x05


#define DEFAULT_RELAY_PIN    '3'

#define PIN_3                '3'

#define PIN_4                '4'


#define PROCESS_CIRCUIT_ID   210

#define PROCESS_MANUAL_SIGNAL 211



/* Global Variable declaration */


TYPE_device* temp_devices[MAX_CIRCUITS * MAX_DEVICES];


TYPE_device* relay_heads[MAX_CIRCUITS];

TYPE_device* current_heads[MAX_CIRCUITS];

TYPE_device* relay_tails[MAX_CIRCUITS];

TYPE_device* current_tails[MAX_CIRCUITS];


TYPE_device* current_on[MAX_CIRCUITS];

TYPE_device* relay_on[MAX_CIRCUITS];


int expectedTime[MAX_CIRCUITS];


int num_devices;

int num_cur;
int ISR_circuit;

char buffer[32];



/* Pin Assignments for LCD */
const int LCD_D0_pin = 2;
const int LCD_D1_pin = 3;
const int LCD_D2_pin = 4;
const int LCD_D3_pin = 5;
const int LCD_RW_pin = 6;
const int LCD_EN_pin = 7;

LiquidCrystal lcd(LCD_EN_pin, LCD_RW_pin, LCD_D3_pin, LCD_D2_pin,
LCD_D1_pin, LCD_D0_pin);

XBee xbee = XBee();

/* Initialize function */
void setup() {
  Serial.begin(9600);
  xbee.begin(9600);
  lcd.begin(20,4);
  setTime(0,0,0,1,1,10);
  num_devices = 0;
  num_cur = 0;

  for (int i=0; i<MAX_CIRCUITS; i++) {
    relay_heads[i] = NULL;
    relay_tails[i] = NULL;
    current_heads[i] = NULL;
    current_tails[i] = NULL;
    current_on[i] = NULL;
    relay_on[i] = NULL;
    expectedTime[i] = 0;
  }

  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, INPUT);
  pinMode(11, INPUT);
  pinMode(12, INPUT);
  pinMode(13, INPUT);
}

void loop() {
  Alarm.delay(0);
  lcd.print("Startup...");
  Alarm.delay(8000);
  node_discover();
  lcd.clear();
  lcd.print("Turning off ALL");
```

```
for (int x=0; x<num_devices; x++) {
  if (temp_devices[x]->name[0] != 'C') {
    /*Alarm.delay(2000);
    lcd.clear();
    lcd.print("Off to ");
    lcd.print(x);*/
    send_signal(temp_devices[x], POWER_OFF, DEFAULT_RELAY_PIN);
  }
}

Alarm.delay(1500);
assign_circuits(0);

Alarm.delay(1500);
assign_circuits(1);

Alarm.delay(1500);
assign_circuits(2);



lcd.clear();
lcd.print(num_devices);
lcd.println(" devices registered");
lcd.print("With ");
lcd.print(num_cur);
lcd.print(" cur devs");
```

```
Alarm.delay(2000);

lcd.clear();



for (int i=0; i<MAX_CIRCUITS; i++) {

  if (relay_heads[i] != NULL) {

    /*lcd.clear();

    lcd.print("Turning on ");

    lcd.print(i);

    Alarm.delay(2000);*/

    send_signal(relay_heads[i], POWER_ON, DEFAULT_RELAY_PIN);

    relay_on[i] = relay_heads[i];


    expectedTime[i] = (now() + TIME_INTERVAL) - 1;

  }

}

Alarm.timerOnce(TIME_INTERVAL,timer_ISR);


for(int y=0; y<Serial.available(); y++) {

  Serial.read();

}


while(1){

  lcd.clear();

  Alarm.delay(0);
```

```
    TYPE_device* man = NULL;

    man = process_response(man, PROCESS_MANUAL_SIGNAL);

    if (man != NULL) {

      manual_device(man);

    }

  }

}




/* Servicing a manual device */

void manual_device(TYPE_device* man) {

  if (current_on[man->circuit_id] == NULL) {

    //lcd.clear();

    //lcd.print("Man device went off");

    if(relay_on[man->circuit_id] != NULL) {

      Alarm.delay(2000);

      send_signal(relay_on[man->circuit_id], POWER_ON,
DEFAULT_RELAY_PIN);

      expectedTime[man->circuit_id] = (expectedTime[man-
>circuit_id] + now()) - 1;

      Alarm.timerOnce(expectedTime[man->circuit_id],timer_ISR);

      Alarm.delay(2000);

      process_response(NULL, 0);

    }

  } else {

    if (current_on[man->circuit_id] != man) {
      //lcd.clear();

      //lcd.print("UH OH");

    } else {

      //lcd.clear();

      //lcd.print("Man device on");

      if(relay_on[man->circuit_id] != NULL) {

        send_signal(relay_on[man->circuit_id], POWER_OFF,
DEFAULT_RELAY_PIN);

        expectedTime[man->circuit_id] = expectedTime[man-
>circuit_id] - now();

      }

    }

  }

  //Alarm.delay(1500);

}



/*Send signals to devices */

void send_signal(TYPE_device *dev, uint8_t msg, char pin){

  if (dev->isLocal == 1) {

    uint8_t frame[12];

    frame[0] = 0x7E; // Frame delimiter

    frame[1] = 0x00; // Size MSB

    frame[2] = 0x05; // Size LSB

    frame[3] = 0x08; // API ID

    frame[4] = 0x01; // Frame ID

    frame[5] = 'D';  // Command D3
    frame[6] = pin;  // ^^
    frame[7] = msg;  // POWER_ON or POWER_OFF
    frame[8] = calculate_checksum(frame, 3, 8);

    Serial.write(frame, 9);

  } else {

    uint8_t frame[20];
    frame[0] = 0x7E;  //Frame Delimiter
    frame[1] = 0x00;  //Frame size MSB
    frame[2] = 0x10;  //Frame size LSB
    frame[3] = 0x17;  //API Identifier
    frame[4] = 0x01;  //Frame ID
    frame[5] = dev->address_high_value[0];
    frame[6] = dev->address_high_value[1];
    frame[7] = dev->address_high_value[2];
    frame[8] = dev->address_high_value[3];
    frame[9] = dev->address_low_value[0];
    frame[10] = dev->address_low_value[1];
    frame[11] = dev->address_low_value[2];
    frame[12] = dev->address_low_value[3];
    frame[13] = 0xFF; //to ignore 16 bit mode
    frame[14] = 0xFE; // ^^
    frame[15] = 0x02; //Execute immediately
    frame[16] = 'D'; // Command D3 (digital IO pin 3)
    frame[17] = pin; // ^^
    frame[18] = msg; // 0x05 = high... 0x04 = low
    frame[19] = calculate_checksum(frame, 3, 19);

    Serial.write(frame, 20);
  }

  Alarm.delay(2000);
  process_response(NULL, 0);
}



void assign_circuits(int i) {
  query_circuit(temp_devices[i]);
  Alarm.delay(2500);
  process_response(temp_devices[i], PROCESS_CIRCUIT_ID);
  Alarm.delay(2500);
  process_response(temp_devices[i], PROCESS_CIRCUIT_ID);
  /*Alarm.delay(1500);
  lcd.clear();
  lcd.println(temp_devices[i]->name);
  lcd.print("Assigned CID: ");
  lcd.print(temp_devices[i]->circuit_id);
  Alarm.delay(2000);*/


  add_device(temp_devices[i]);
}
```

```
void add_device(TYPE_device* dev) {
  int circuit = 0;
  if (dev->circuit_id != NULL) {
    circuit = dev->circuit_id;
  }

  if ((dev->name[0] == 'C') || (dev->name[0] == 'c')) {
    if ((dev->name[1] == 'U') || (dev->name[1] == 'u')) {
      if ((dev->name[2] == 'R') || (dev->name[2] == 'r')) {
        /*Alarm.delay(1000);
        lcd.clear();
        lcd.print("Adding current to ");
        lcd.print(circuit);
        Alarm.delay(2000);*/
        if (current_tails[circuit] == NULL) {
          current_heads[circuit] = dev;
          current_tails[circuit] = dev;
          current_tails[circuit]->prev = NULL;
        } else {
          current_tails[circuit]->prev = dev;
          current_tails[circuit] = dev;
          current_tails[circuit]->prev = NULL;
        }
      }
    }
  } else {
    /*Alarm.delay(1000);
    lcd.clear();
    lcd.print("Adding relay to ");
    lcd.print(circuit);
    Alarm.delay(2000);*/
    if (relay_tails[circuit] == NULL) {
      relay_heads[circuit] = dev;
      relay_tails[circuit] = dev;
      relay_tails[circuit]->prev = NULL;
    } else {
      relay_tails[circuit]->prev = dev;
      relay_tails[circuit] = dev;
      relay_tails[circuit]->prev = NULL;
    }
  }
}


void query_circuit(TYPE_device* dev) {
  if (dev->isLocal == 1) {
    uint8_t frame[12];
    frame[0] = 0x7E; // Frame delimiter
    frame[1] = 0x00; // Size MSB
    frame[2] = 0x05; // Size LSB
    frame[3] = 0x08; // API ID
    frame[4] = 0x01; // Frame ID
    frame[5] = 'I';  // Command D3
    frame[6] = 'S';  // ^^
    frame[7] = 0x01;  // POWER_ON or POWER_OFF
    frame[8] = calculate_checksum(frame, 3, 8);

    Serial.write(frame, 9);

    Alarm.delay(5000);
    //process_response(dev, PROCESS_CIRCUIT_ID);
  } else {
    uint8_t frame[19];
    frame[0] = 0x7E;  //Frame Delimiter
    frame[1] = 0x00;  //Frame size MSB
    frame[2] = 0x10;  //Frame size LSB
    frame[3] = 0x17;  //API Identifier
    frame[4] = 0x01;  //Frame ID
    frame[5] = dev->address_high_value[0];
    frame[6] = dev->address_high_value[1];
    frame[7] = dev->address_high_value[2];
    frame[8] = dev->address_high_value[3];
    frame[9] = dev->address_low_value[0];
    frame[10] = dev->address_low_value[1];
    frame[11] = dev->address_low_value[2];
    frame[12] = dev->address_low_value[3];
    frame[13] = 0xFF; //to ignore 16 bit mode
    frame[14] = 0xFE; // ^^
    frame[15] = 0x02; //Execute immediately
    frame[16] = 'I'; // Command D3 (digital IO pin 3)
    frame[17] = 'S'; // ^^

    frame[18] = 0x01;

    frame[19] = calculate_checksum(frame, 3, 19);


    Serial.write(frame, 20);

  }

  Alarm.delay(5000);

  //process_response(dev, PROCESS_CIRCUIT_ID);

}

void node_discover() {

  /* Register a device that is the local device */


  TYPE_device* local_temp_device =
(TYPE_device*)malloc(sizeof(TYPE_device));

  local_temp_device->isLocal = 1;

  local_temp_device->circuit_id = 0;

  local_temp_device->name[0] = 'L';

  local_temp_device->name[1] = 'O';

  local_temp_device->name[2] = 'C';

  local_temp_device->name[3] = '\0';

  temp_devices[0] = local_temp_device;

  num_devices = 1;


  Serial.print("+++");

  readReply(1);

  Alarm.delay(1000);
  Serial.print("ATND\r");
  int r = 1;
  int cur_sensor = 0;

  while (r > 0) {
    cur_sensor = 0;
    Alarm.delay(5000);
    r = readReply(2);
    if (r == 0) {
      break;
    }

    TYPE_device* temp_device =
(TYPE_device*)malloc(sizeof(TYPE_device));

    temp_device->source[0] = buffer[0];
    temp_device->source[1] = buffer[1];
    temp_device->source[2] = buffer[2];
    temp_device->source[3] = buffer[3];
    convert(temp_device->source, temp_device->source_value, 4);

    temp_device->address_high[0] = '0';
    temp_device->address_high[1] = '0';
    temp_device->address_high[2] = buffer[4];
    temp_device->address_high[3] = buffer[5];
    temp_device->address_high[4] = buffer[6];
    temp_device->address_high[5] = buffer[7];
    temp_device->address_high[6] = buffer[8];

    temp_device->address_high[7] = buffer[9];

    convert(temp_device->address_high, temp_device-
>address_high_value, 8);


    temp_device->address_low[0] = buffer[10];

    temp_device->address_low[1] = buffer[11];

    temp_device->address_low[2] = buffer[12];

    temp_device->address_low[3] = buffer[13];

    temp_device->address_low[4] = buffer[14];

    temp_device->address_low[5] = buffer[15];

    temp_device->address_low[6] = buffer[16];

    temp_device->address_low[7] = buffer[17];

    convert(temp_device->address_low, temp_device-
>address_low_value, 8);


    temp_device->signalStrength[0] = buffer[18];

    temp_device->signalStrength[1] = buffer[19];

    convert(temp_device->signalStrength, &(temp_device-
>signalStrength_value), 2);


    temp_device->name[0] = buffer[20];

    temp_device->name[1] = buffer[21];

    temp_device->name[2] = buffer[22];

    temp_device->name[3] = '\0';


    temp_device->isLocal = 0;

    temp_device->circuit_id = 0;
```

```
  temp_device->prev = NULL;

  if ((buffer[20] == 'C') || (buffer[20] == 'c')) {
    if ((buffer[21] == 'U') || (buffer[21] == 'u')) {
      if ((buffer[22] == 'R') || (buffer[22] == 'r')) {
        cur_sensor = 1;
        num_cur++;
      }
    }
  } else {
    temp_device->name[0] = 'R';
    temp_device->name[1] = 'E';
    temp_device->name[2] = 'L';
    temp_device->name[3] = '\0';
  }

  temp_devices[num_devices] = temp_device;
  num_devices++;
}

Alarm.delay(2000);
Serial.print("ATCN\r");
Alarm.delay(2000);

for(int y=0; y<Serial.available(); y++) {
  Serial.read();
}
```

```
}


/*
 * Read Reply
 *
 * Used only for Node Discovery
 *   ND is performed in AT command mode and the reply
 *   must be read in one char at a time
 *
 */
int readReply(int max_returns) {
  char c;
  unsigned long time;
  unsigned long start = millis();
  int i=0;
  int num_returns = 0;
  memset (buffer,'\0',32);

  while (true) {
    if(Serial.available()) {
      c = Serial.read();

      if (c == '\r') {
        num_returns += 1;
        if (num_returns >= max_returns || i==0) {
          return i;
        }
      } else {
        num_returns = 0;
        buffer[i++] = c;
      }


    } else {
      time = millis();
      if ((time-start) > 10000) {
        lcd.print("No reply");
        return 0;
      }
    }
  }
}


/* Timer ISR
 *
 * The controller uses a time share based schedule giving each of
the devices even amounts of time
 * The ISR is called when the time interval has elapsed
 * The device currently on is given a powerOFF signal and the
next device is given powerON
 * This ISR also takes care of removing the device at the head of
the queue and moving it to the end
 *
 * Performs:
```

```
 *  1 - Power off the device currently on

 *  2 - Move the currently on device from the head of the queue
to the rear

 *  3 - Delay for some number of milliseconds

 *  4 - Power on the device at the head of the queue

 *  5 - Set the new currentlyOn device

 *

 */


void timer_ISR() {

  /*lcd.clear();

  lcd.print("Entered TimerISR");

  Alarm.delay(2000);

  lcd.clear();*/

  TYPE_device* temp_device;

  for (int i=0; i<MAX_CIRCUITS; i++) {

    if (current_on[i] == NULL) {

      if (now() >= expectedTime[i] && expectedTime[i] != 0) {

        if (relay_heads[i] != relay_tails[i]) {

          send_signal(relay_on[i], POWER_OFF, DEFAULT_RELAY_PIN);

          lcd.print(relay_on[i]->name);

          lcd.println(":OFF");


          //Move the head device to the tail

          temp_device = relay_heads[i];

          relay_heads[i] = relay_heads[i]->prev;

          relay_tails[i]->prev = temp_device;
          temp_device->prev = NULL;

          relay_tails[i] = temp_device;


          //Delay

          Alarm.delay(TIME_DELAY_ONOFF);


          send_signal(relay_heads[i], POWER_ON,
DEFAULT_RELAY_PIN);

          lcd.setCursor(0,1);

          lcd.print(relay_heads[i]->name);

          lcd.println(":ON");


          relay_on[i] = relay_heads[i];


        } else {

          lcd.print("None/One device");

          Alarm.delay(1500);

          if (relay_heads[i] != NULL) {

            //send_signal(relay_heads[i], POWER_ON,
DEFAULT_RELAY_PIN);

            expectedTime[i] = (now() + TIME_INTERVAL) - 1;

          }

        }

      } else {

         //if (expectedTime[i] != 0) {

           //Alarm.timerOnce((expectedTime[i] - now()),
timer_ISR);

         //}
      }
    } else {
      //lcd.print("Manual device on");
      //Alarm.delay(1500);
    }
  }
  //reset the time interval
  Alarm.timerOnce(TIME_INTERVAL,timer_ISR);
}


void convert(char *values, uint8_t *output, int length) {
  uint8_t bytesread = 0;
  uint8_t tempbyte = 0;
  uint8_t val = 0;
  int i = 0;

  while (bytesread < length) {
    val = values[i++];
    if (val <='9') {
      val = val - '0';
    } else {
      val = 10 + val - 'A';
    }
    if ((bytesread & 1) == 1) {
      output[bytesread >> 1] = (val | (tempbyte << 4));
    } else {

      tempbyte = val;

    }


    bytesread++;

  }

}


uint8_t calculate_checksum(uint8_t* frame, int start, int finish)
{

  int sum = 0;

  for (int i=start; i<finish; i++) {

    sum = sum + frame[i];

  }


  sum = sum & 0xFF;


  sum = 0xFF - sum;

  return (uint8_t)sum;

}






TYPE_device* process_response(TYPE_device* dev, int toProcess){

  Rx64IoSampleResponse ioSample = Rx64IoSampleResponse();
  xbee.readPacket();

  /*lcd.clear();

  Alarm.delay(1500);

  lcd.print("Packet Read");

  Alarm.delay(1500);

  lcd.clear();*/

  if (xbee.getResponse().isAvailable()) {

    if (xbee.getResponse().getApiId() == RX_64_IO_RESPONSE) {

      xbee.getResponse().getRx64IoSampleResponse(ioSample);

      if ((ioSample.getSampleSize() == 1) &&
(ioSample.containsDigital())) {

        uint8_t sample[8];

        for(int i=0; i<=8; i++) {

          sample[i] = ioSample.isDigitalOn(i, 1);

        }


        uint8_t val = (sample[0]) | (sample[1] << 1) | (sample[2]
<< 2);

        int circuit_id = 0;

        switch (val) {

          case 0x07:

            circuit_id = 7;

            break;

          case 0x06:

            circuit_id = 6;

            break;

          case 0x05:
```

```
    circuit_id = 5;
    break;
  case 0x04:
    circuit_id = 4;
    break;
  case 0x03:
    circuit_id = 3;
    break;
  case 0x02:
    circuit_id = 2;
    break;
  case 0x01:
    circuit_id = 1;
    break;
  case 0x00:
    circuit_id = 0;
    break;
  default:
    circuit_id = 0;
    break;
}

if (toProcess == PROCESS_CIRCUIT_ID) {
  dev->circuit_id = circuit_id;
  /*lcd.clear();
  lcd.print(dev->circuit_id);
  lcd.setCursor(0,1);
```

```
          lcd.print(val, HEX);

          Alarm.delay(3000);*/

        }


        if (toProcess == PROCESS_MANUAL_SIGNAL) {

          /*Alarm.delay(2000);

          lcd.print("man sig CID: ");

          lcd.print(circuit_id);

          Alarm.delay(2000);*/

          TYPE_device* check = current_heads[circuit_id];


          uint32_t response_low;

          uint32_t address_low;

          while (check != NULL) {

            response_low = 
ioSample.getRemoteAddress64().getLsb();

            address_low = check->address_low_value[0];

            address_low = address_low << 8;

            address_low = address_low | check-
>address_low_value[1];

            address_low = address_low << 8;

            address_low = address_low | check-
>address_low_value[2];

            address_low = address_low << 8;

            address_low = address_low | check-
>address_low_value[3];


            /*Alarm.delay(3000);
            lcd.clear();

            lcd.print(response_low);

            lcd.setCursor(0,1);

            lcd.print(address_low);*/


            if (address_low == response_low) {

              if (sample[4] == 0x00) { //Device ON

                current_on[circuit_id] = check;

              } else { //Device OFF

                current_on[circuit_id] = NULL;

              }


              return check;

            } else {

              check = check->prev;

            }

          }

        }

      }

    }

    else {

      lcd.clear();

      if (xbee.getResponse().getApiId() ==
REMOTE_AT_COMMAND_RESPONSE) {

        RemoteAtCommandResponse RATCR =
RemoteAtCommandResponse();

        xbee.getResponse().getRemoteAtCommandResponse(RATCR);
```

```
          if (RATCR.isOk()) {

            if (RATCR.getValueLength() > 0) {

              if (toProcess == PROCESS_CIRCUIT_ID) {

                uint8_t val =
RATCR.getValue()[RATCR.getValueLength()-1];

                val = val & 0x07;

                int circuit_id = 0;

                switch (val) {

                  case 0x07:

                    circuit_id = 7;

                    break;

                  case 0x06:

                    circuit_id = 6;

                    break;

                  case 0x05:

                    circuit_id = 5;

                    break;

                  case 0x04:

                    circuit_id = 4;

                    break;

                  case 0x03:

                    circuit_id = 3;

                    break;

                  case 0x02:

                    circuit_id = 2;

                    break;
                  case 0x01:
                    circuit_id = 1;
                    break;
                  case 0x00:
                    circuit_id = 0;
                    break;
                  default:
                    circuit_id = 0;
                    break;
                }
                if (toProcess == PROCESS_CIRCUIT_ID) {
                  dev->circuit_id = circuit_id;
                  return dev;
                }
              }
            }
          }
        }
        else if(xbee.getResponse().getApiId() ==
  AT_COMMAND_RESPONSE) {
          AtCommandResponse ATCR = AtCommandResponse();
          xbee.getResponse().getAtCommandResponse(ATCR);

          if (ATCR.isOk()) {
            if (ATCR.getValueLength() > 0) {
              if (toProcess == PROCESS_CIRCUIT_ID) {
                uint8_t val =
  ATCR.getValue()[ATCR.getValueLength()-1];

                val = val & 0x0;


                switch (val) {

                  case 0x07:

                    dev->circuit_id = 7;

                    break;

                  case 0x06:

                    dev->circuit_id = 6;

                    break;

                  case 0x05:

                    dev->circuit_id = 5;

                    break;

                  case 0x04:

                    dev->circuit_id = 4;

                    break;

                  case 0x03:

                    dev->circuit_id = 3;

                    break;

                  case 0x02:

                    dev->circuit_id = 2;

                    break;

                  case 0x01:

                    dev->circuit_id = 1;

                    break;

                  case 0x00:
```

```
                dev->circuit_id = 0;
                break;
              default:
                dev->circuit_id = 0;
                break;
            }
            Alarm.delay(2000);
            return dev;
          }
        }
      }
    } else {
      lcd.print("Expected IO Sample, got ");
      lcd.print(xbee.getResponse().getApiId(), HEX);
      Alarm.delay(3000);
      return NULL;
    }
  }
}
else if (xbee.getResponse().isError()) {
  lcd.clear();
  lcd.print(xbee.getResponse().getErrorCode());
  Alarm.delay(3000);
  return NULL;
}
return NULL;
```